# Süsteemprogrammeerimine keeles

Loeng 3

*milles lisaks esimese labori kirjeldamisele räägitakse, mis on Stream, kuidas näidatakse vigu, mis on standardlibrad, räägime programmi kontrollvoost, uuesti konstantidest koos makrodega ja veidi preprotsessorist*

# Eelmises osas (printf)

- printf() esimene argument on string, teised argumendid lähtuvad stringi sisust

```
printf("%d = %o oct = %X hex\n", x, x, x);
```

  10 = 12 oct = A hex

- %[-][laius].[täpsus][h/l][tüüp]
  - %d – integer (digit)
  - %f – float
  - %s – string
  - %c – char

# Eelmises osas (Konstandid)

- Preprotsessori konstandi defineerimine:

  #define *nimi väärtus*

  #define PI 3.14

# Eelmises osas (Täisarvude jagamine)

- 3/2 = 1 (Jagatise täisosa)
- 3%2 = 1 (Jagatise jääk)

# Eelmises osas (Tüübiteisendused)

- Explicit casting (Teisendamine vägisi)
  - (tüüp)muutujanimi
  - (float)i
- Implicit casting (Teisendamine automaatselt)
  - kahe eriti muutja jagamisel on vastus võimsamast tüübist:
    long double > double > float > long > int > short
  - märgiga ja märgita arvu teisendamine toimub märgita arvuks (vältimiseks võrdlustel teisendame nt floatideks)

# Eelmises osas (Char)

- Konstandina 'a' 'b' '1' '$'
- Olemuselt number: kasutage nt tehetes
- Numbriväärtused saab ASCII tabelist
- Erimärgid:

  \n – reavahetus  \0 – null

  \' – '  \" - "
- Lugemine/kirjutamine
  - **int** getchar(void)
  - int putchar(char)

# Eelmises osas (Võrdlemine ja loogika)

- Võrdlemine:

  `a==b, a!=b, a<b, a>b, a<=b, a>=b`

- Loogikatehted:

  *not* !

  *or* ||

  *and* &&

- Vastus kas 1 (tõene) või 0

# Eelmises osas (Massiivid)

- Deklareerimine:

  *tüüp nimi[suurus];*

  ```
  int a[10]; /* massiiv 10 täisarvust */
  ```

- Indeks algab 0-st
- Algväärtustamine:

  ```
  int a[3] = {12, 14, 16};
  int a[2][3] = {{1, 2, 3}, {3, 2, 1}};
  ```

- Massiivi elemendi n aadress:
  - addr = baas + n * sizeof(tüüp)

# Eelmises osas (Pointerid)

- Pointer – muutuja mis hoiab mõne muu muutuja aadressi
- float *floatpointer;
- "Address of" & (võtab aadressi)
- "Value of" * (võtab viidatud aadressilt)
- Massiivi puhul on tegemist pointeriga massiivi esimesele elemendile
- Pointeraritmeetika:
  - `int a[10]; int * pa = a + 1; /* pa = a[1] */`
- Pointer pointerile: int **intp_p;

# Tänane kava

- *Laboratoorne töö 1*
- Pointerid, jm.
- Failide kasutamine
- Veahaldus
- Librade kasutamine
- Juhtvoog (Control flow)
- Konstandid ja Makrod
- Preprotsessor

# Pointerid ja massiivid

- Pointerite ja massiivide vahel on tugev ja oluline seos.
- Defineerime:

  ```
  int a[10];
  ```

- a tüübiks määratakse automaatselt int * (pointer int-ile) ja tema väärtus on &a[0] (kõige esimese massiivi elemendi aadress)
- Järeldus:

  `int *pa = a;` on pärast int a[10] määramist lubatud ja töötav rida.

# Tehted pointeritega

- Pointer on mäluaadress: mingi number
  - pointeri maht mälus sõltub arhitektuurist
- "Mingi numbriga" on võimalik teha ka tehteid:

```
int a[10];
int *pa=a;      /* pa on a[0] aadress */
int *pb= a+1;   /* pb on a[1] aadress */
*(pb+2)=3;      /* a[3] on nüüd 3;*/
```

# Kahe massiivi vahetamine

- Vahetame kaks massiivi

```
int a1[10], a2[10];
int *karud = a1, *rebased = a2, *temp;
/* nüüd viitavad pa1 ja a1 samasse kohta:
 karud[2] == a1[2], *(karud+3) == a1[3], jne.
  */

/* massiivide vahetamiseks vahetame viited */
temp = karud; karud=rebased; rebased=temp;

/*nüüd viitab karud massiivi a2 elementidele*/
/* ja rebased massiivi a1 elementidele */
```

# Pointer pointerile

- Kuna pointer on ise ka muutuja võime võtta ka pointeri aadressi
- Süntaks:

```
int **ptr_to_ptr;
int *ptr;
int i = 1;

ptr_to_ptr = &ptr;   /* ptr_to_ptr väärtus on ptr aadress*/
ptr = &i;            /* ptr väärtus on nüüd i aadress */
*ptr_to_ptr = &i;    /* sama mis eelmine */
*(*ptr_to_ptr) = 2;  /* määra i väärtuseks 2 */
```

- Pointerite juurde tuleme tagasi

# Funktsiooni argumendid – väärtus versus viide

- Igast funktsiooniargumendist tehakse koopia, mis hävib kui funktsioon töö lõpetab
- Kui funktsioon peab oma argumendi sisu muutma, tuleb funktsioonile anda pointer vastavale kohale mälus.

# Funkstiooni argumendid – väärtus versus viide

```
void test(int koopia, int *pointer)
{
koopia = 1;    /* muutub kohalik koopia*/
*pointer = 1;  /* muutub muutuja ise */
}

main()
{
int i1 = 0, i2 = 0;
/* i1 antakse väärtusega, i2 aadressiviitena*/
test (i1,&i2);
/* i1 ei muutu ja i2 muutub */
}
```

# Stringid

- Stringide defineerimine stringikonstantide abil:

```
char nimi[8] = "Raibert";
char nimi[] = "Raibert";
```

- Definitsioonipõhiselt on tegemist tähemärkide massiiviga
  - Stringikonstantidele lisandub otsa '\0' märk. Seega on stringi *nimi* pikkus 4. Kolm tähte + '\0' terminaator
- Nii saab initsialiseerida vaid tähemärkide massiive.
- Stringide töötlusfunktsioonid on failis <strings.h>
- Stringidest pikemalt edaspidi

# Funktsiooni main() argumendid

- Funktsioon main() saab "välismaailmast" parameetreid. Parameetrite tüüp on ettemääratud:

```
main (int argc, char ** argv) {…}
```

  - argc – argumentstringide arv
  - argv – argumentstringide massiiv;
    **NB!** argv[0] on alati programminimi
- Programmi käivitamine "*minuproge arg1 arg2*"
  - argc = 3
  - argv[0] = "minuproge"
  - argv[1] = "arg1"
  - argv[2] = "arg2"

# Prindime välja programmi argumendid

```
#include <stdio.h>

/*
prindib programmile antud argumendid
*/

main (int argc, char **argv)
{
   while (argc--) { printf("%s\n", *(argv++)); }
}
```

# Stream

- Mis on *stream*?
- Standardsed *stream*id: stdin, stdout, stderr
- Streame saab ka ise luua: FILE * fp;
- FILE on andmetüüp, mis on defineeritud <stdio.h> failis ja seega peab iga *stream*e kasutav programm sisaldama rida:

```
#include <stdio.h>
```

# Mida *stream*iga teha saab?

- Avada
- Kasutada (= lugeda, kirjutada)
- Sulgeda

# *Stream*i avamine

- FILE *fopen(char * filename, char *mode);

  mode

  "r" ava fail lugemiseks

  "w" loo või ava fail kirjutamiseks, fail tühjendatakse

  "a" *append*; loob või avab faili selle lõppu kirjutamiseks

  "r+" ava fail täiendamiseks (st lugemine & kirjutamine)

  "w+" ava fail täiendamiseks, fail tühjendatakse

  "a+" *append*; avab faili täiendamiseks, kirjutamine lõppu

- Vea korral tagastatakse NULL (0)

```
FILE * stream1 = fopen("/etc/passwd", "r");
```

# *Stream* ja sisend/väljund

printf(), scanf(), getchar(), putchar(), gets(), puts() töötavad stdin ja stdout peal.

- Vastavad funktsioonid suvalistele streamidele:

  fprintf(), fscanf(), getc(), putc(), fgets(), fputs() võtavad esimeseks argumendiks streami:

```
int printf(const char* format, ...)
int fprintf(FILE* stream, const char* format, ...)
```

- Järgnevad programmiread on ekvivalentsed:

```
printf("test\n");
fprintf(stdout, "test\n");
```

# *Stream*i sulgemine

- fclose(FILE * steam);

```
fclose(fp);
```

- Oluline! *Stream* tuleb pärast kasutamist sulgeda.

# Standardsed *stream*id

- Eeldefineeritud *stream*e ei pea programmeerija avama/sulgema
- stdin – scanf(), getchar(), gets()

  ümbersuunamine: `programm < sisendfail`
- stdout – printf(), putchar(), puts()

  ümbersuunamine: `programm > väljundfail`
- Kui on vaja ümber suunata nii stdin kui stdout, kasuta:

  ```
  programm < sisendfail > väljundfail
  ```
- **NB!** Puhverdamine toimub reakaupa

# Stderr

- Kolmas *stream*, mis avatakse iga programmi puhul.
- Kasutatakse veateadete näitamiseks. Neid näidatakse ekraanil isegi siis, kui standardväljund on suunatud (st `programm > väljund`)
- Ümbersuunamine on võimalik:

```
programm 2> error.log
```

```
if ( (fp=fopen(“infile”,”r”)) == NULL) {
  fprintf(stderr,”Cannot open file\n”);
  exit(-1);
}
```

# Librad

- Tihtikasutatavate funktsioonide jaoks on osad funktsioonid ette ära defineeritud.
- C omab mitut standardset *libraryt*
- Librast funktsiooni kasutamiseks tuleb lisada vastav päis (*header*) ning kompileerimisel libra sisse linkida

```
#include <math.h>
   ...
main() { printf("2 kuubis on %d\n", pow(2, 3)); }
```

- Kui ülaltoodud näites #include puuduks, trükiks funktsioon vastuseks midagi juhuslikku

# Standardsed librad

- Matemaatika `#include <math.h>`
- Stringid `#include <string.h>`
- Sisend/Väljund `#include <stdio.h>`
- Dünaamiline mäluhaldus `#include <stdlib.h>`

# Juhtvoog (Tingimuslik väärtustamine)

- Programmirida võib oma väärtuse saada sõltuvalt mingi tingimuse kehtimisest:
- Süntaks:

```
tingimus ? väärtus_kui_tõene : väärtus_kui_väär
```

- Näide:

```
int x = y > z ? y : z;
```

samaväärne if-lausega:

```
 if (y > z)
      x = y;
  else
      x = z;
```

# Eelnev ja järgnev ++ ja --

- ++i kõigepealt suurendab i, ja siis tagastab (suurendatud) väärtuse
- i++ tagastab kõigepealt väärtuse ja siis suurendab i
- Näide

```
int i=0, j=0;
printf("i=%d, j=%d\n", ++i, j++);
```

```
i=1, j=0
```

- Eraldiseisvalt on ++i ja i++ ekvivalentsed.